(1)

Organizing Plan Libraries in Subsumption Hierarchies: Specificity Based Plan Selection

Vibhu Mittal[†] **Cecile Paris** **Ramesh Patil** **William Swartout**[†]

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
U.S.A.

{mittal,paris,ramesh,swartout}@ISI.EDU
Phone: +1 (310) 822-1511
Fax: +1 (310) 823-6714

Abstract:

As the number of plans in a plan library grows, the importance of selecting a plan efficiently also increases. Previous planners have not addressed this issue in great detail, because they typically had only tens of plans -- as domain models grow more and more specialized, and planners incorporate new macro-operators through learning, plan libraries can become more than a magnitude greater in size. This paper addresses the issue of organizing plan libraries in specificity based hierarchies. In particular, we look at how a plan library can be represented in a frame-based KL-ONE style system. Such systems offer powerful mechanisms to handle dynamically changing knowledge bases. In particular, the subsumption classifier mechanism in such systems, offers a simple and efficient means of indexing new plans and goals into the hierarchy. We illustrate the feasibility of this approach by using examples from an implemented system, and describe some of the other advantages that accrue from the use of such a framework for organizing large plan libraries.

The authors are listed in alphabetical order and gratefully acknowledge the support of NASA-Ames grant NCC 2-520 and DARPA contract DABT63-91-C-0025.

[†]Also affiliated with the Department of Computer Science, University of Southern California, Los Angeles, CA 90089.

1 Introduction

Planning systems may vary widely in their control strategy or their application domain, but they have all had a *plan library*, a resource which stores and organizes the plans for use by the planner. As domain models grow more specialized, and as planners incorporate new macro-operators through learning, plan libraries can become very larger. In such situations, it becomes very important to organize the plans in a structure designed for efficient indexing and access. This issue was not raised by previous planners such as NOAH [7], TWEAK [1] and STRIPS [2], because their plan libraries were relatively small, and consisted of operators that were relatively independent of each other.

In addition to the question of efficient access, another criterion that the organization of the library must consider is the expression of relationships between the different plans. This is essential, because, as we will show in the paper, the knowledge about the inter-relationships between the plans is critical for plan selection, when a number of plans are applicable, as well as for goal reformulation, when there are no plans that match the goal exactly. Schemes such as a hash-table look-up mechanism, therefore, are not suitable, because, while they offer a fast access and indexing of the plans, they do not offer any means of annotating inter-relationships among them.

In this paper, we describe one approach to the organization and representation of plan libraries: our system uses the mechanisms in a frame-based, inheritance network knowledge representation (KR) system to both store and index plans in a type specific hierarchy. The use of such a system to implement plan retrieval and selection requires that the plan-capabilities, and the goals, be expressed in terms in the KR language. We describe in the following sections, how specificity based plan selection can be implemented within a KR framework, and the issues that arise in mapping plans and goals to expressions that can be used by such a KR system during classification. We illustrate our approach with examples and conclude with a discussion on its strengths and weaknesses.

2 A Specificity Based Hierarchical Organization of the Plans

In our framework,[1] we view planning as a means of generating action sequences for a wide range of problem solving tasks, including diagnosis, critiquing and scheduling. Plans in our system resemble macro-operators in some planning systems, such as STRIPS, NOAH and TWEAK -- they are ordered sequences of actions, which can be applied in certain contexts, if all the given pre-conditions are satisfied. However, our approach is not limited to specific types of plans or planning systems. We are addressing the issues that arise when there are a large number of plans, many of them being variants for achieving similar goals in differing situations. For instance, there can be a number of plans to diagnose a decserver: one to diagnose *any* decserver, indicated by the following specification:

```
(DIAGNOSE (OBJECT (D is (A DECSERVER))))
```

another to diagnose decservers which are connected to source-computers, indicated by:

```
(DIAGNOSE (OBJECT (D is
        (A DECSERVER (CONNECTED-TO SOURCE-COMPUTER)))))
```

and yet another to diagnose a decserver connected to another source-computer directly through a link (i.e., not through a lanbridge):

```
(DIAGNOSE (OBJECT (D is
        (A DECSERVER (LINE-CONNECTED SOURCE-COMPUTER)))))
```

Given a goal to diagnose a decserver, it is important for the planner be able to select the most appropriate plan in the situation. Large plan libraries containing numerous plans like the ones mentioned above, have not been very common in the past, but with more and more systems integrating learning with planning, the situation is fast changing. The need for organizing plan libraries becomes evident when viewed in this context.

[1]The Explainable Expert Systems (EES) Framework [6, 8].

There are two major requirements in the organization of plan libraries:

1. The ability to efficiently index plans, given a particular goal, and retrieve the most specific plan for that goal, taking into account all the information available, and

2. The ability to express plan inter-relationships.

The ability to index plans efficiently further depends upon two factors:

1. *What* the system considers during the match: most planners, such as PRODIGY [?] for instance, match plans based only on the EFFECT specification, i.e., the action and its arguments. These arguments are typically either variables or constants, and not domain dependent concept restrictions. This operation results in a set of possible plans. This set of plans is then examined, serially, to see if the constraints are satisfied. A plan is then selected from the set remaining after inapplicable ones have been removed. This is typically done on the basis of control knowledge, using selection strategies, such as "pick the most specific," "pick the most general,", etc.

2. *How* the plans are organized in a plan library: previous planners have adopted various approaches to organizing plans, based on their perspective of how important efficient plan matching is to the system's performance. Thus, for instance, since the major source of computational complexity in planning has typically been in the sequencing, a large number of planners have encoded their plan libraries in a flat, serial fashion, thus forcing the examination of each and every plan for every given goal. Earlier versions of PRODIGY, for instance, did this. Other approaches have also been used: a hash table, mapping goals to applicable plans (recent versions of PRODIGY); the use of discrimination nets, as in NOAH; etc.

Few planners have addressed these two issues of organizing libraries based on both plan matching and representing inter-relationships. Yet, these issues are important if a good organization for large plan libraries is to be devised. Each of these issues interact and influence one another -- for instance, since plan selection is based on matching goals with EFFECTs *and* the CONSTRAINTS, the matching process could be made more efficient by combining the two sources of information and using it all to find applicable plans. However, there planners have not attempted to integrate the two. The information being used in the match process affects the organization of the plans in the library; however, the need to be able to represent plan relationships, also constrains the organization scheme, and therefore, the type of indexing schemes possible. For instance, hash tables, while offering fast access, would not be able to satisfy the requirement of representing plan inter-relationships. Surprisingly, plan inter-relationships have rarely been considered during plan organization, inspite of some significant benefits: consider for instance the three plans mentioned above: it can be seen that the second plan and the third plan are both specializations of the first one, because they are both plans that diagnose decservers in specialized situations. The third plan is also a specialization of the second one, *if* the knowledge base specifies that the relation LINE-CONNECTED is a specialization of the more general relation CONNECTED-TO. Should the last plan mentioned above fail to achieve a goal, the next one to be attempted should be the second one, rather than the first one. However, if the relationships between plan 1, plan 2 and plan 3 are not represented, the planner would not be able to select plan 2 over plan 1 (It would consider them both equally applicable). Thus, the organization should reflect some of the control knowledge that is usually specified in a non-transparent, procedural fashion in planners.

One possible method of organizing the plans is in the form of a hierarchy: given a specificity ordering between plans, it is an easy matter to organize them in a hierarchy. Well designed hierarchies offer a reasonably fast and efficient means of indexing plans; should a goal fail to match any plans exactly, it is still possible to find plans that are more specific and more general than the one required. These plans can be then used to reformulate the goal in terms of the plans that the system does possess.

However, hierarchies are not very easy to modify, especially if new terms or relationships are defined in terms of one another, or if new plans are added or deleted. Thus, the use of static discrimination networks, such as those used in NOAH, to index the plan library, is not satisfactory, though it does offer some advantages over the other approaches that have been used.

In the following section, we show how the use of a inheritance-based, frame classification system such as LOOM [4,5] allows our system to build *dynamic* hierarchies based on *type specificity*. Such a hierarchical organization provides fast and efficient indexing into the plan library; the classification capability coupled with the underlying knowledge representation, allows the dynamic modification of the terms and their relationships in the knowledge base.

3 Organizing Plan Libraries as Concept Hierarchies

From the preceding discussion, it is clear that there is a need to organize large plan libraries into specificity based hierarchies. As we mentioned earlier, plans in our system resemble macro-operators in other systems. They have a number of components, of which the relevant ones are:[2]

- the :CAPABILITY slot, which indicates the sort of goals the plan is capable of achieving;
- the :PRECONDITIONS slot, specifying the features that must be true in the state of the world for that plan to be instantiated and executed (preconditions in our system do not cause other goals to be posted).
- and a :RESULTS slot, which specifies the type of the result of plan execution -- for instance, a plan to determine whether a component had power or not, would have BOOLEAN as its result specification; a plan to find all instances of source-computers that were connected to a given decserver, would have (SET-OF COMPUTER) as its result specification.

[2]There are other components, such as the :METHOD or the body of the plan, which we will not elaborate upon in this paper.

```
(DEFINE-PLAN Diagnose-Component
   :CAPABILITY (diagnose (object (C is (a component))))
   :RESULTS (a potential-problem)
   :METHOD (
            [ ... code for the plan ... ]
           )
)
```

Figure 1: A sample plan to diagnose a component.

An example of a plan in our domain of diagnosing decservers is shown in Figure 1.

It has no preconditions and its result is a `potential-problem`. It should be mentioned here that our plan language allows the expression of some types of constraints as part of the :CAPABILITY specification, in addition to the ones in the :PRECONDITIONS slot.

To be able to represent plans such as these in a frame based KR system, the system must be able to map the :CAPABILITY, :PRECONDITIONS and the :RESULTS slots appropriately into expressions in the KR language. This mapping must be designed in such a way that specializations of a plan result in corresponding KR expressions that are specializations of the expression representing the more general plan. In other words, the plan lattice should be mapped into an isomorphic concept lattice. There are many possible transformations to ensure this mapping. We shall now describe briefly the transformations for plan capabilities and goals in our system.

3.1 Mapping Plan Capabilities to Concept Expressions

The mapping between plan capabilities and concept expressions can be easily derived by representing the *semantics* of the plan's capability description. This is especially true in our system, where the plan language allows the expression of various constraints as part of the :CAPABILITY specification. Consider for instance, the following plan :CAPABILITY specification:[3] (DIAGNOSE (OBJECT (D is (A DECSERVER)))) In this, the simplest case, the capability description specifies that the plan is capable of diagnosing D, a decserver. The KR language used in our system is LOOM [4, 5], one of the KL-ONE family of languages. One possible definition for a corresponding LOOM concept of the above capability description is:

```
(DEFCONCEPT DIAGNOSE-DECSERVER
   :IS (:AND DIAGNOSE (:THE OBJECT DECSERVER)))
```

The expression above defines a new concept DIAGNOSE-DECSERVER. Diagnose-Decserver is an *action* of type DIAGNOSE. It therefore inherits all the roles and restrictions defined on DIAGNOSE. It also has one relation restriction defined on it. This restriction is defined on the relation OBJECT: there can be only *one* relation named OBJECT on an instance (this is the keyword :the), and the range of this relation (the domain is diagnose-decserver) is restricted to the type DECSERVER.

Consider now the following capability description:

```
(DIAGNOSE (OBJECT (D is (A DECSERVER (CONNECTED-TO SOURCE-COMPUTER)))))
```

In this case, the capability description specifies that the plan is capable of diagnosing decservers which are connected to source-computers. This is a specialized version of the plan discussed previously -- the one to diagnose decservers in general -- and if the resulting LOOM concept is to form an isomorphic lattice structure, it is essential that the LOOM transformation be such that the corresponding LOOM concept for this plan is subsumed by the concept definition for the first one. What is needed in this case, is a concept, also of type DIAGNOSE (since this is also a diagnosing action), but the OBJECT under diagnosis is another concept: a complex concept, as yet unnamed, which represents a decserver connected-to a source-computer. This can be represented in LOOM as follows:

```
(DEFCONCEPT DIAGNOSE-DECSERVER-CONNECTED-TO-SOURCE-COMPUTER
   :IS (:AND  DIAGNOSE
          (:THE OBJECT (:AND DECSERVER
                             (:SOME CONNECTED-TO SOURCE-COMPUTER)))))
```

[3]In the more traditional operator representations, the capability could be represented as:(:CAPABILITY (DIAGNOSE ?D)) (:CONSTRAINTS (TYPE-P ?D DECSERVER)).

The LOOM definition states that the concept is of type DIAGNOSE, whose only OBJECT role has its range restricted to a concept: a decserver, one of whose connected-to roles[4], is filled by a source-computer. The concept defined above is a specialization of the previous one, since the range restriction is a specialization of decserver.

These examples illustrate the basic principles behind the mappings from plan capabilities to LOOM concept expressions.

3.2 Mapping Goals to Concept Expressions

In order to do the match a goal to the plans, goals must also be transformed into concept expressions that can be used in indexing the plan hierarchy. This can be done in a fashion similar to that for plan capabilities. There is however, an additional complication in transforming goals to LOOM expressions, because goals have *instances* as well as concepts in them. For instance, if the system were to be given a goal of the form (DIAGNOSE (OBJECT DECSERVER-123)), the desired behavior of the system would be to find and return the most specific plan applicable for the instance DECSERVER-123. The most specific plan can be found efficiently if the plans are organized in a type specific hierarchy, by converting the goal into a LOOM form and using that to index into the plan hierarchy.

KL-ONE style classifiers require *concept expressions* for classification -- they cannot classify instances -- and therefore it becomes essential that the mixed-mode expression that the goal would normally be converted to (following the transformation sequence described for plan capabilities) be coerced into a form suitable for use by the classifier. There are two ways of coercing the mixed-mode concept into a form that can be used by LOOM: one, is to go 'down', and convert the concept expression to a 'pure' instance, or to go 'up' and transform it into a pure concept expression. A discussion on the differences between these two approaches is beyond the scope of this paper. In this section, we describe how goals can be mapped into 'pure' concept expressions, which can be used to index the plan hierarchy for applicable plans.

It is important to keep in mind that during this transformation

[4]This is specified by the keyword :some

of a goal to a LOOM expression, and then to a LOOM concept expression, no *usable information* is dropped in the mapping process. This concept of usable information can be better explained with an example: Consider, for instance, the goal of diagnosing the instance `DECSERVER-123` that were posted above. If all that were known about `DECSERVER-123` was of type `decserver`, the system should have retrieved the `DIAGNOSE-DECSERVER` plan; on the other hand, if it was known that `DECSERVER-123` was related to an instance of `source-computer` through the relation CONNECTED-TO, then the plan `DIAGNOSE-DECSERVER-CONNECTED-TO-SOURCE-COMPUTER` should have been the most specific one returned. Note that the retrieval of the second plan is still based on the same syntactic goal being posted; the extra knowledge about the connection of `DECSERVER-B` is available and retrieved from the knowledge base, but is not explicitly stated in the goal. Plan selection strategies based on static discrimination nets, that did not take into account (for instance) all sub-types of the relation CONNECTED-TO, would not be able to correctly index plans that involved those specializations. This can easily occur, if for instance, the knowledge base is modified even slightly after the discrimination network is constructed. In a system such as ours, new concept descriptions are created by various resources; in any learning system, the possibility of new terms being defined due to learning cannot be ruled out. It is thus important for any system that the 'discrimination network' be able to take into account the dynamic nature of the knowledge base and the underlying concept hierarchy, in terms of which the plans are likely to be indexed.

This brings us back to the original issue, that of coercing the mixed-mode expressions into forms that the classifier is capable of handling. One possible solution is to transform the goals not into concepts with instances as role fillers, but rather to find the *most specific concept type* (MST) of each instance in the goal, and use these types to construct the corresponding LOOM expression. This approach has one very important implication -- classifiers in KL-ONE style languages do not classify instances because the definitional component of information cannot be circumscribed -- by finding the MST of an instance, the reasoning is now being restricted to only consider *named expressions* -- concepts that exist (either because they were user-defined, or defined by the system as a result of a conjunctive expression appearing in a plan capability), in the classifier hierarchy. Without this restriction, it would not have been possible for the classifier to work; in essence, the system requests the classifier to reason using only concepts it already knows about.

Consider for instance, the example given below:

```
(tell (adder instance-1))
(retrieve ?x (instance--type instance-1 ?x)) ==> (|C|ADDER)
```

In the first case, retrieving the `instance--type` of `instance-1` returns the concept ADDER.

```
(tell (multiplier instance-2))
(tell (connected-to instance-1 instance-2))
```

A relationship between `instance-1` and `instance-2` through the relation `connected-to` is asserted. The MST that the system would return in response to the query that previously returned `(|C| ADDER)`, depends upon any other concept compositions that the system knows about. If there were no concepts that had an adder connected to a multiplier in the domain model, the system would return an `(|C| ADDER)`, as in the case above. However, if there were to be a concept in the system that related an `adder` and a `multiplier` via a `connected-to` relation, the query would return that concept, even if it were an internally-defined concept. In our case, the system returned the concept `Concept_243`, where `Concept_243` was defined to be:

```
     (DEFCONCEPT CONCEPT_243
         :IS (ADDER (:SOME CONNECTED-TO MULTIPLIER))
         :ATTRIBUTES (:SYSTEM-DEFINED))
```

`CONCEPT_243` was defined by the system automatically, when the following plan capability was defined in LOOM:

```
     (<ACTION> ... (<role-name> (A is (A (ADDER (CONNECTED-TO MULTIPLIER))):
```

This is because, the way this is transformed into LOOM resulted in the following expression:

```
     (DEFCONCEPT ADCG-592
        :IS (:AND <ACTION> (:THE <ROLE-NAME>
                 (:and ADDER (:SOME CONNECTED-TO MULTIPLIER)))))
```

When LOOM is given this definition, it is forced to create an internally named concept (in our case, `concept_243`). Thus, the underlying concept hierarchy in terms of which the plans can be indexed, contains not only the terms that the user defines, but also system-defined terms such as the one above, which arise due to the complex plan definitions. This enables the system to retrieve the correct plan from the plan hierarchy by taking into consideration *all* incidental information about the instance *that it can take advantage of*. Thus, the plans known to the system automatically determine the information about the goal instance that will be used in indexing the plans, in addition to the user defined *type*, which is what a plan selection mechanism based on static discrimination networks would be limited to.

A brief description of our plan matching algorithm is shown in Figure 2.

4 Advantages of this Approach

There are many advantages of using an approach which combines the power and flexibility of frame-based classifier system with a planning system to organize and index plan libraries in an efficient manner. As systems grow and add more specialized plans and refine their domain model, the problem of indexing plan libraries is likely to become larger. Previous approaches to this problem have not considered the use of such a powerful mechanism, because they were either implemented as isolated stand-alone planning systems, or the number of plan operators was small enough for this issue to not be raised. There are a number of advantages in the use of such a framework. Some of these are:

1. An efficient organization of plans in a hierarchy, leading to fast efficient access. The hierarchy closely integrated with the domain model, and is maintained by the classifier dynamically. Changes in the domain model cause both the domain relationships as well as the plan hierarchy to be updated accordingly. In contrast, static discrimination networks, based on one version of the domain model, can be crippled by the use of a slightly different domain model that invalidate some of the previous relationships. Changes in the plan capability statements, as well as the constraints, are

For each plan:

1. Parse each plan and collect and combine all its `:preconditions` with its `:capability` specification.
2. Map this new `:capability` specification to a corresponding LOOM expression (these translation routines will depend upon the grammar of the plan language and the KR language).
3. Generate a LOOM concept whose definition consists of the above expression and is subsumed by a concept that stands for plans.
4. Classify the new plan concept in the plan hierarchy.

For each goal:

1. Parse the posted goal and find all the instances in the goal.
2. For each instance in the goal posted, find the most specific concept type that subsumes it.
3. Replace the actual instances in the posted goal with their types.
4. Transform this modified goal (containing only concepts) to a LOOM expression in a fashion similar to the one for plan capabilities.
5. Using this LOOM expression, find where it fits into the T-Box hierarchy of the domain knowledge. Retrieve exact matches, immediate ancestors and descendants and return those that correspond to plan-capabilities.

Figure 2: The Plan Selection Algorithm

also taken into account by the classifier and do not result in a need to make appropriate changes in different parts of the system.

2. The system can take advantage of incidental information about goal instances: if an instance happens to have properties for which a plan exists, then the plan will be returned by the plan selection mechanism (This is due to the fact that the classifier checks for every property on the instance in terms of named-concept-expressions, as was discussed earlier).

3. Some types of goal reformulations are greatly facilitated by such a framework. The hierarchical organization of plan libraries, combined with its close coupling with the underlying domain model, facilitates at least two types of goal reformulations:

(a) Consider for instance, a goal such as `(DIAGNOSE DECSERVER-313)`, where `DECSERVER-313` happens to be an instance of `decserver`. If there were no plans in the system to diagnose decservers, most planners would return failure (A few recent systems, such as SOAR [3] would attempt to use *weak methods* to solve the goal, but that might not be the best approach -- since their approach would be to try and attempt the diagnosis from first principles). For instance, in the case of DECSERVER, the KB might specify the fact that it has two disjoint sub-types: DECSERVER-200 and DECSERVER-500. The system can then check to see whether it possesses plans that can diagnose these two types of decservers. If it can, then an attempt can be made to diagnose the given instance, by executing both the plans and trying to combine their results. This type of goal reformulation, based on knowledge about a concept, available in the domain model, and a hierarchical classification of plans based on type, is greatly facilitated by a framework that combines the two.

(b) Another possibility of goal failure arises when the system is unable to find an appropriate plan, inspite of its existence, because of a difference in the terms used in the goal and the plan. For instance, the goal could be of the form: `(DIAGNOSE SOURCE-COMPUTER-411)`, where the goal instance is of type SOURCE-COMPUTER. There may be no plans in the knowledge base to diagnose SOURCE-COMPUTERs; this may be perhaps due to the fact that any computer, capable of supplying particular software versions to a decserver can be its SOURCE-COMPUTER. Thus the plan to diagnose computers can actually be applied to a source-computer, by taking into account its connectivity and its software versions. Since information such as this can be expressed in the form of :IMPLIES relations in these systems, the classifier will automatically be able to retrieve `DIAGNOSE-COMPUTER` as a potential candidate for this instance.

5 Conclusions

Many systems use a planner as one of their problem solving components. Most of these planners, however, are not well integrated with the rest of the system. Plans are matched with goals by using the plan :CAPABILITIES (without taking into account the constraints as well), using different mechanisms such as table look-up, discrimination networks, etc. Planners have traditionally had their own mechanisms for plan storage and retrieval, independent of the rest of the system components. Plans have typically had limited access to the knowledge base, through accessor functions as part of their constraints. In this paper, we have described one approach towards integrating the storage and selection mechanisms of a planner with the underlying knowledge framework, and using a subsumption classifier to retrieve plans by transforming goals appropriately.

Anadvantage of this approach lies in its ability to facilitate goal reformulations. Failure to find an exact match for a goal should result in the system replanning the goal based on other plans in its knowledge base rather than goal failure. One way to reformulate a goal efficiently is by restructuring the goal in terms of plans that the system does possess, using information in the knowledge base to help achieve its task. A plan hierarchy such as ours is very useful in this regard, because it takes into consideration features such as disjoint coverings and inheritance relationships while classifying the plans. Finding plans that are 'close' to the desired match is a simple matter in this case.

Our approach is not without its some disadvantages however, chief among which is the requirement of a KR system that does classification. Since the classifier uses its knowledge about the relationships between terms in the domain model for classification, it is necessary that plan selection criteria be expressible in the KR language. In our system, since type specificity was the desired criterion, there was no need for additional specification. In cases of *plan recognition* for instance, a mapping between the body of the plan (or rather the the structure that is being used for recognition, such as for instance, the kernel, or the justification structure), and an appropriate KR expression must be derived. This is not difficult, however, and can lead to a better representation of the plan structure -- one that is closely integrated with the representation of the domain model -- in a single, uniform fashion.

References

[1] David Chapman. Planning for Conjunctive Goals. *Artificial Intelligence*, 32:333–377, 1987.

[2] Richard E. Fikes and Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 88 – 98. Morgan Kaufmann Publishers, Inc., San Mateo, CA., 1990.

[3] John E. Laird, Allen Newell, and Paul S. Rosenbloom. SOAR: An architecture for general Intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[4] Robert MacGregor. A Deductive Pattern Matcher. In *Proceedings of the 1988 Conference on Artificial Intelligence*, St Paul, Mn, August 1988. American Association of Artificial Intelligence.

[5] Robert MacGregor. The evolving technology of classification-based knowledge representation systems. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, San Mateo, California, 1991.

[6] Robert Neches, William Roy Swartout, and Johanna Doris Moore. Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Transactions on Software Engineering*, SE-11(11):1337--1351, November 1985.

[7] Earl D. Sacerdoti. *A Structure for Plans and Behaviour*. Elsevier-North Holland, 1977.

[8] William R. Swartout and Stephen W. Smoliar. On making expert systems more like experts. *Expert Systems*, 4(3):196 – 207, August 1987.